lib/main/power_unittest.ath

```
load "power"

#------------------------------------------------------------------------
define M1 := no-renaming

assert (theory-axioms Monoid.theory)

(!prove-property Monoid.Power.right-plus M1 Monoid.theory)

(!prove-property Monoid.Power.left-neutral M1 Monoid.theory)

(!prove-property Monoid.Power.right-one M1 Monoid.theory)
(!prove-property Monoid.Power.right-two M1 Monoid.theory)
(!prove-property Monoid.Power.right-times M1 Monoid.theory)

assert (get-property Abelian-Monoid.commutative M1 Abelian-Monoid.theory)

(!prove-property Abelian-Monoid.Power-left-times M1 Abelian-Monoid.theory)

#------------------------------------------------------------------------
load "nat-power"

define M1 := (renaming |{Monoid.+* := N.**, Monoid.+ := N.*,
                         Monoid.<0> := N.one}|)

define [n x] := [?n:N ?x:'T]

(!claim (forall n x . (N.** x (S n)) = x N.* (N.** x n)))

(!prove-property Monoid.Power.right-plus M1 Monoid.theory)

(!prove-property Monoid.Power.left-neutral M1 Monoid.theory)

(!prove-property Monoid.Power.right-one M1 Monoid.theory)

(!prove-property Monoid.Power.right-two M1 Monoid.theory)

(!prove-property Monoid.Power.right-times M1 Monoid.theory)

(!prove-property Abelian-Monoid.Power-left-times M1 Abelian-Monoid.theory)

#------------------------------------------------------------------------
declare **: (T) [T N] -> T [400]

define M1 := (renaming |{Monoid.+* := **}|)

assert (M1 (theory-axioms MM.theory))

(!prove-property Monoid.Power.right-plus M1 MM.theory)

(!prove-property Monoid.Power.left-neutral M1 MM.theory)

(!prove-property Monoid.Power.right-one M1 MM.theory)

(!prove-property Monoid.Power.right-two M1 MM.theory)

(!prove-property Monoid.Power.right-times M1 MM.theory)

(assert (get-property Abelian-Monoid.commutative M1 MAM.theory))

(!prove-property Abelian-Monoid.Power-left-times M1 MAM.theory)

#------------------------------------------------------------------------
load "list-of.ath"

declare Join*: (T) [(List T) N] -> (List T)

```

```
68  define M1 := (renaming |{Monoid.+* := Join*, Monoid.+ := List.join,
69                           Monoid.<0> := nil}|)
70
71  # Define Join* as an instance of **
72
73  assert (map lambda (P) (M1 P)
74              [Monoid.Power.right-zero Monoid.Power.right-nonzero])
75
76  (!prove-property Monoid.Power.right-plus M1 Monoid.theory)
77
78  (!prove-property Monoid.Power.left-neutral M1 Monoid.theory)
79
80  (!prove-property Monoid.Power.right-one M1 Monoid.theory)
81
82  (!prove-property Monoid.Power.right-two M1 Monoid.theory)
83
84  (!prove-property Monoid.Power.right-times M1 Monoid.theory)
85
86  # List.join isn't commutative, so we don't have Monoid.Power-left-times.
87
88  #-------------------------------------------------------------------------
89  open Monoid
90
91  define L1 := (1 :: 2 :: nil)
92
93  define three := (S N.two)
94
95  #(set-debug-mode "rewriting")
96
97  let {adapter := (renaming |{+* := Join*, + := List.join, <0> := nil}|);
98       [get prove chain chain-> chain<-] := (proof-tools adapter Monoid.theory);
99       [+ <0> +*] := (adapter [+ <0> +*]);
100      _ := (!prove Power.right-two)}
101    let {_ := (!claim Power.right-nonzero)}
102    (!chain [(L1 +* three)
103             --> (L1 + (L1 +* N.two))      [Power.right-nonzero]
104             --> (L1 + L1 + L1)            [Power.right-two]])
105
106 (eval (L1 List.join L1 List.join L1))
```